Relatório 2º Projeto ASA 2023/2024

Grupo: TP033

Aluno(s): João Gomes (106204) e Sofia Piteira (106194)

Descrição do Problema e da Solução:

O nosso programa oferece uma solução para encontrar o maior percurso num grafo direcionado, independentemente da presença ou ausência de Componentes Fortemente Conexos. A nossa abordagem para lidar com o problema inclui a identificação de Componentes Fortemente Conexos e a transformação do grafo original, num sem as mesmas. Este passo permite a posterior aplicação de um algoritmo destinado a encontrar o caminho mais longo em um grafo direcionado acíclico.

O código começa por ler um grafo direcionado através de pares de vértices e, em seguida, realiza uma busca em profundidade (DFS) para encontrar as Componentes Fortemente Conexas (SCCs) do grafo. Identificadas as SCCs, constrói um Grafo Acíclico Direcionado (DAG) condensado, representando cada SCC como um nó. Ao percorrer os nós do DAG por ordem inversa, calcula o comprimento do caminho mais longo até cada nó, atualizando os valores. O resultado final é o comprimento máximo do caminho no grafo original.

Análise Teórica:

- Leitura dos dados de entrada, (N) número de nós, (M) número de conexões e arestas do grafo e criação das listas de adjacências - função readinput()

  Complexidade O (M): Porque como todas as operações dentro do for loop têm complexidade constante a complexidade da função é determinada pelo número de iterações do loop.

- Implementação do algoritmo DFS para encontrar os Componentes Fortemente Conexos (SCCs) de um grafo direcionado - função findSCCs()

Complexidade O (N + M): Porque o for loop exterior percorre todos os N vértices do grafo e o interior é percorrido no máximo M vezes.

- Aplicação do algoritmo DFS, com o objetivo de encontra os vértices por ordem de saída do grafo - função dfs()

Complexidade O (N + M): Porque o for loop exterior percorre todos os N vértices do grafo e o interior é percorrido no máximo M vezes.

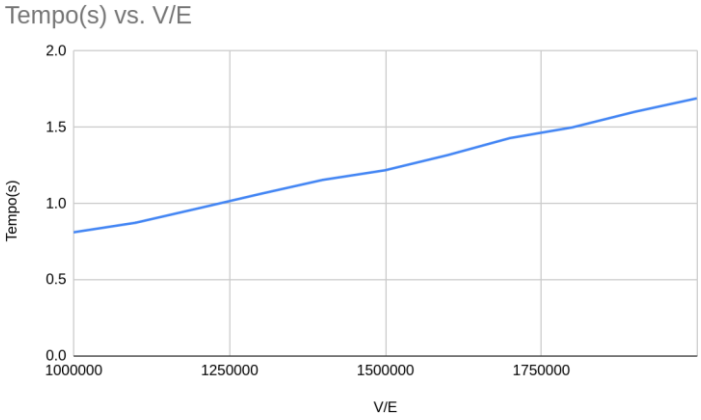
- Construção de um grafo direcionado acíclico DAG a partir do grafo original, representando-o com uma lista de adjacências e excluindo as arestas que conectam vértices da mesma SCC - função builtDag()

Complexidade O (N + M): Porque o for loop exterior percorre todos os N vértices do grafo e o interior é percorrido no total um máximo de M vezes.

- Cálculo do caminho mais longo em um DAG, ao iterar esta lista de adjacências percorremos o grafo e guardamos numa variável o valor máximo de arestas percorridas - função longhestPath()

Complexidade O (N + M): Porque o for loop exterior percorre todos os N vértices do grafo e o interior é percorrido no total um máximo de M vezes.

A complexidade total da solução é **O (N + M)** em que o N é o número de vértices do grafo e o M o número de arestas.



V/E	Tempo(s)
1000000	0.809
1100000	0.872
1200000	0.966
1300000	1.061
1400000	1.153
1500000	1.216
1600000	1.315
1700000	1.427
1800000	1.497
1900000	1.599
2000000	1.688

O gráfico acima representa os testes executados pelo nosso código. Para uniformizar os nossos testes utilizámos o mesmo valor de pessoas e ligações, sendo esse o número representado no eixo horizontal. No eixo vertical encontram-se os tempos de execução para os inputs utilizados no decorrer dos testes.

Como podemos observar, se N = M, então o crescimento é 2N, logo, **linear**. Tal como se verifica no gráfico.

Com base nestes dados, concluímos que a complexidade do código é N + M, oque vai de encontro com oque concluímos na nossa análise teórica.